

---

# Large-Scale Transient Sensitivity Analysis of a Radiation-Damaged Bipolar Junction Transistor via AD

Eric T. Phipps, Roscoe A. Bartlett, David M. Gay, and Robert J. Hoekstra

Sandia National Laboratories, Albuquerque NM 87185, USA <sup>†</sup>

**Summary.** Automatic differentiation (AD) is useful in transient sensitivity analysis of a computational simulation of a bipolar junction transistor subject to radiation damage. We used forward-mode AD, implemented in a new Trilinos package called Sacado, to compute analytic derivatives for implicit time integration and forward sensitivity analysis. Sacado addresses element-based simulation codes written in C++ and works well with forward sensitivity analysis as implemented in the Trilinos time-integration package Rythmos. The forward sensitivity calculation is significantly more efficient and robust than finite differencing.

**Key words:** Sensitivity analysis, radiation damage, bipolar junction transistor, forward mode, Trilinos, Sacado, Rythmos

## 1 Introduction

One of the primary missions of Sandia National Laboratories is certifying the safety, security, and operational reliability of the USA's nuclear weapons stockpile. An important aspect of this mission is qualifying weapon electronic circuits for use in abnormal (e.g., fire) and hostile (e.g., radioactive) environments. In the absence of underground testing and with the decommissioning of fast pulse neutron test facilities such as the Sandia Pulsed Reactor (SPR), emphasis has been placed on using computational modeling and simulation as a primary means for electrical system qualification. To further this objective, Sandia has been developing computer codes to simulate individual semiconductor devices and electronic circuits subject to damage resulting from radioactive environments. In semiconductor devices, this radiation damage creates displaced "defect" species that can move through the device, capture and release electronic charge, and undergo reactions. Modeling of this defect physics introduces many uncertain parameters into the computational model, and calibrating the model with existing experimental data reduces the uncertainty in these parameters. In this paper we discuss transient parameter sensitivity analysis of a bipolar junction transistor (BJT) subject

---

<sup>†</sup> Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000. This document is released as SAND2007-7767C.

to radiation damage. The computed sensitivities provide information needed for a derivative-based optimization method to calibrate the model, and also give detailed analysis of the radiation damage mechanisms and their relative importance to device performance metrics, to guide future model improvements. The semiconductor device and radiation defect physics are implemented in a large-scale finite element code called Charon, developed at Sandia, which uses the Trilinos solver collection [8] for linear solvers, preconditioners, nonlinear solvers, optimization, time integration, and automatic differentiation. Transient sensitivities are computed using a forward sensitivity method implemented in the Trilinos time integration package Rythmos, with state and parameter derivatives computed via automatic differentiation using the Trilinos package Sacado.

Much of the foundation for this work has been discussed previously [4], where our approach for computing derivatives in large-scale element-based applications like Charon was presented. In that paper we discussed the implementation details and performance of computing state Jacobians and Jacobian-transpose products on a simple convection diffusion problem, using the C++ AD tools Fad [3] and Rad [6]. Here we report on the application of that approach to the full radiation defect semiconductor device physics model implemented in Charon and extend it to include parameter derivatives, observation functions and transient sensitivities. In Sect. 2, we review the element-level approach for computing derivatives in large-scale applications. The automatic differentiation tools Fad and Rad have been incorporated into a new AD package called Sacado and have become part of Trilinos. We discuss this package in more detail in Sect. 3. The transient sensitivity approach as implemented in the Trilinos package Rythmos is discussed in Sect. 4, and the radiation defect physics for the bipolar junction transistor is presented in Sect. 5. Finally, we discuss the transient sensitivity analysis of the BJT in Sect. 6, comparing the overall performance of the approach to a black-box style finite difference method. We found the intrusive approach using AD and forward transient sensitivities to be significantly more efficient and robust than the finite-difference approach. The Trilinos packages discussed here, including Sacado and Rythmos, are available in Trilinos 8.0 [1].

## 2 Differentiating Element-Based Models

Here we provide a brief overview of the approach for computing derivatives of element-based models published previously [4]. In general we are interested in models that (possibly after some spatial discretization) can be represented as a large system of differential algebraic equations

$$\begin{aligned} f(\dot{x}, x, p, t) &= 0, \\ \hat{g}(p, t) &= g(\dot{x}(t), x(t), p, t), \end{aligned} \quad 0 \leq t \leq T \quad (1)$$

where  $t \in \mathbb{R}$  is time,  $x, \dot{x} \in \mathbb{R}^n$  are the state variables and their time derivatives,  $p \in \mathbb{R}^m$  are model parameters and  $g : \mathbb{R}^{2n+m+1} \rightarrow \mathbb{R}^l$  is one or more observation functions. Typically we refer to  $f : \mathbb{R}^{2n+m+1} \rightarrow \mathbb{R}^n$  as the global residual and  $\hat{g}$  as the reduced observation. For the purposes of this paper, we think of  $n$  as possibly very large, on the order of millions, while  $m$  is reasonably small, on the order of 100 and  $l$  is on the order of 1 to 10. For element-based models,  $f$  can be decomposed as the sum

$$f(\dot{x}, x, p, t) = \sum_{i=1}^N Q_i^T e_{k_i}(P_i \dot{x}, P_i x, p, t) \quad (2)$$

over a large number of elements  $N$  taken from a small set  $\{e_k\}$  of element functions  $e_k : \mathbb{R}^{2n_k+m+1} \rightarrow \mathbb{R}^{n_k}$  where each  $n_k$  is at most a few hundred. Here we are using the term “ele-

ment” in a generic sense not restricted to finite-element models. The matrices  $P_i \in \mathbb{R}^{n_{k_i} \times n}$  and  $Q_i \in \mathbb{R}^{n_{k_i} \times n}$  map global vectors to the local element domain and range spaces respectively. Typically  $g$  has a similar decomposition. As discussed in [4], for systems that are a spatial discretization of a set of PDEs, one must distinguish between interior elements that are decomposed as above and boundary elements that have some other set of boundary conditions applied. The extension of (2) to include boundary conditions is straightforward and will not be treated here. For implicit time integration and transient sensitivity analysis, one must compute the following derivatives, which have corresponding decompositions into element derivatives:

$$\alpha \frac{\partial f}{\partial \dot{x}} + \beta \frac{\partial f}{\partial x} = \sum_{i=1}^N Q_i^T \left( \alpha \frac{\partial e_{k_i}}{\partial \dot{x}} + \beta \frac{\partial e_{k_i}}{\partial x} \right) P_i, \quad \frac{\partial f}{\partial p} = \sum_{i=1}^N Q_i^T \frac{\partial e_{k_i}}{\partial p} \quad (3)$$

for given scalars  $\alpha$  and  $\beta$ . As discussed in [4], computing the element derivatives in (3) is well suited to automatic differentiation because they involve relatively few independent and dependent variables, do not involve a large number of operations, and do not require parallel communication. Moreover the complexity of the AD calculation is independent of the number of elements.

### 3 Automatic Differentiation with Sacado

In previous work [4], the feasibility and efficiency of computing the element derivatives (3) in the C++ finite-element simulation code Charon using the AD tools Fad [3] and Rad [6] was discussed. Since that work, we have made AD tools based on Fad and Rad into a new package called Sacado that is now part of the Trilinos collection. This package provides operator overloading for forward, reverse, and Taylor mode automatic differentiation in C++ codes. The forward mode tools are based on Fad and use expression templates for efficiency, but have been completely redesigned to support a more flexible software design and conformance to the C++ standard. The new tools use the same interface as Fad, allowing drop-in replacement for Sacado. The reverse mode tools are essentially a repackaging of the original Rad, but also provide enhanced debugging modes and better support for passive variables (variables which are really constants but are declared to be an AD type, see [4] for why these are a nuisance for Rad). The Taylor mode is a simple but efficient univariate Taylor polynomial implementation that uses handles instead of expression templates. All tools are templated to permit nesting AD types for computing higher derivatives.

As discussed in [4], our approach for applying these AD types to application codes is to template the C++ code that computes the element functions  $e_k$  and to instantiate this templated code on the AD types. At the start of each element computation for a given derivative calculation, a preprocess operator is used to map the global solution vectors  $x$  and  $\dot{x}$  to the local element space ( $P$  mapping from Eq. (2)) and initialize the corresponding AD type for the independent variables. Then the template instantiation of the element function for this AD type is called to compute the element derivative. Finally a post-process operator extracts the derivative values (from either independent or dependent variables depending on the AD type) and sums them into the global derivative objects ( $Q$  mapping). The manual part of the differentiation process is contained within these preprocess and post-process operators, and new operators must be defined each time new AD types are added to the code. However the physics and its finite element discretization is contained within the templated element functions  $e_k$ , and therefore the process of differentiating new physics is completely automatic.

Ideally the interface between the pre/post-process operators and element functions would be the only place in the code where templated code must be called from non-templated code, but in practice there are numerous such places. To encapsulate this interface and facilitate easy addition of new AD types, a template manager and iterator are provided by Sacado to store the different instantiations of templated application code classes and loop over them in a type-independent way. The ideas of template meta-programming [2] are used to implement this cleanly. Also, analysis tools such as sensitivity computations and optimization require an application code interface to set, retrieve, and compute derivatives with respect to parameters. However, application codes rarely provide such an interface and therefore Sacado provides a simple parameter library class to facilitate computing parameter derivatives by AD.

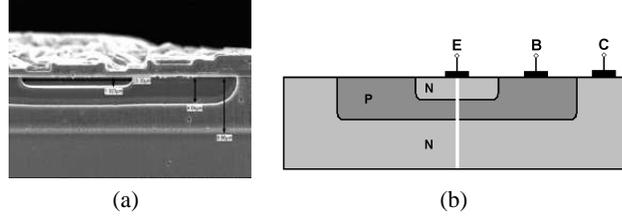
All of these tools have been incorporated into Charon to enable computation of first and second derivatives with respect to both state variables and parameters. As discussed in [4], this approach is highly intrusive to the application code and has required significant software engineering to incorporate into Charon. While complicated and certainly not black-box, we have found this approach highly effective for computing derivatives in large-scale, parallel, evolving physics application codes, both in terms of the computational cost of the derivative calculations [4] and the human time required to develop and maintain the code. Since incorporating Sacado into a large-scale application code is as much (if not more) about the software engineering to support the templating than the AD itself, Sacado provides a small one dimensional finite element application called FEApp to demonstrate these tools and techniques.

## 4 Transient Sensitivity Analysis with Rythmos

The Rythmos package in Trilinos implements selected explicit and implicit time integration solvers based on the IDA package [9]. In this study, we employed a variable-order, variable step-size backward-difference time integrator (BDF) to solve the initial value state equations (1) and the forward sensitivity problem

$$\begin{aligned} \frac{\partial f}{\partial \dot{x}} \left( \frac{\partial \dot{x}}{\partial p} \right) + \frac{\partial f}{\partial x} \left( \frac{\partial x}{\partial p} \right) + \frac{\partial f}{\partial p} &= 0, \\ \frac{\partial \hat{g}}{\partial p} &= \frac{\partial g}{\partial \dot{x}} \frac{\partial \dot{x}}{\partial p} + \frac{\partial g}{\partial x} \frac{\partial x}{\partial p} + \frac{\partial g}{\partial p}, \end{aligned} \quad 0 \leq t \leq T \quad (4)$$

given appropriate initial conditions. Rythmos uses a highly modular object-oriented infrastructure based on the abstract numerical algorithm approach of Thyra [1], where the sensitivity equations in (4) are formulated as a single implicit ODE and solved using a stepper class that also solves the forward state equations (1). A small amount of coordinating code is used to efficiently implement the *staggered corrector* forward sensitivity method [5], where each (non-linear) state time step is solved to completion before the (linear) sensitivity time step equation is solved for the update to the sensitivities  $\partial x/\partial p$ . The observation function  $g$  and the reduced sensitivity  $\partial \hat{g}/\partial p$  are then computed at the end of each time step using an observer subclass. An error control scheme based on local truncation error estimates is employed to control errors on the states  $x$ , but error control for the sensitivities  $\partial x/\partial p$  is not currently implemented (in the future this limitation will be removed). The Trilinos package NOX [1] solves the implicit BDF time step equations, and numerous direct and iterative linear solvers and preconditioners provided by Trilinos can be used to solve the resulting linear systems of equations through a single abstract interface provided by the Trilinos package Stratimikos [1]. Finally, the Sacado AD classes are used to efficiently provide accurate partial derivatives  $\partial f/\partial \dot{x}$ ,  $\partial f/\partial x$ ,  $\partial f/\partial p$ ,  $\partial g/\partial \dot{x}$ ,  $\partial g/\partial x$ , and  $\partial g/\partial p$  for the Rythmos forward state and sensitivity solver code.



**Fig. 1.** Scanning electron microscope image of an NPN BJT (a) and diagram of the emitter (E), base (B), and collector (C) regions (b). The simulation domain is a  $9 \times 0.1$  micron slice (white vertical line) below the emitter contact with contacts at each end and a contact embedded in the strip representing the base contact.

## 5 Radiation Defect Semiconductor Device Physics

We are interested in applying the transient sensitivity analysis technique discussed in the previous section to computational models of semiconductor devices subject to radiation damage. In this section we provide a brief description of the radiation defect semiconductor device physics implemented in the physics code Charon developed at Sandia, applied to an NPN bipolar junction transistor (BJT) shown in Fig. 1(a). Modeling this physics is quite detailed and due to space constraints not all aspects of the model nor its implementation in Charon are discussed (more details can be found in [7]). As shown in Fig. 1(b), a BJT is a device with three electrical contacts referred to as the emitter (E), base (B), and collector (C). Each contact is attached to the boundary of a region of the device where the silicon lattice has been modified by the introduction of impurities to produce an abundance of free electrons ( $N$ -doping) in the emitter and collector regions or holes ( $P$ -doping) in the base region [12]. Charged carriers (electrons and holes) flow through the device as dictated by the electric field in the body and the electric potential or carrier flux prescribed at the contacts. When a device is exposed to a radiation environment, the radiation interacts with the device's lattice material and may “knock out” an atom within the lattice, leaving a vacancy (a void) and an interstitial (free material atom), referred to as a Frenkel pair. These vacancies and interstitials (collectively referred to as defect species) can carry charge, move throughout the device, and interact through various reactions such as capture/release of electrons/holes and recombination. The diffusion and transport of carriers and defect species are governed by the following partial differential equations [12]:

$$-\nabla \cdot (\lambda^2 \nabla \psi) = \left( p - n + C \sum_{i=1}^N Z_i Y_i \right) \quad (5)$$

$$\nabla \cdot (-\mu_n n \nabla \psi + D_n \nabla n) = \frac{\partial n}{\partial t} + R_n \quad (6)$$

$$\nabla \cdot (\mu_p p \nabla \psi + D_p \nabla p) = \frac{\partial p}{\partial t} + R_p \quad (7)$$

$$\nabla \cdot (\mu_{Y_i} Y_i \nabla \psi + D_{Y_i} \nabla Y_i) = \frac{\partial Y_i}{\partial t} + R_{Y_i}, \quad i = 1, \dots, N \quad (8)$$

where  $\psi$  is the scalar electric potential,  $n$  and  $p$  are the electron and hole concentrations,  $Y_i$  is the concentration of defect species  $i$  for  $i = 1, \dots, N$ ,  $Z_i$  is the integer charge number of defect species  $i$ ,  $C$  is the static doping profile,  $\lambda$  is the minimal Debye length of the device,  $R_x$  is

**Table 1.** Sample of the 84 defect reactions and corresponding parameters. Column # refers to the parameter number in Fig. 2. Superscripts denote charge states,  $V$  refers to a vacancy,  $BV$  to a boron-vacancy complex,  $PV$  to a phosphorous-vacancy complex,  $\sigma$  to the reaction cross-section and  $\Delta E$  to the reaction activation energy.

#	Reaction	Parameter	Value	#	Reaction	Parameter	Value
13	$e^- + V^- \rightarrow V^{--}$	$\sigma$	3.0e-16	46	$e^- + PV^0 \rightarrow PV^0$	$\sigma$	1.5e-15
14	$V^{--} \rightarrow e^- + V^-$	$\Delta E$	0.09	79	$h^+ + V^- \rightarrow V^0$	$\sigma$	3.0e-13
15	$V^{--} \rightarrow e^- + V^-$	$\sigma$	3.0e-16	83	$V^+ \rightarrow h^+ + V^0$	$\Delta E$	0.05
16	$e^- + V^0 \rightarrow V^-$	$\sigma$	2.4e-14	83	$V^+ \rightarrow h^+ + V^0$	$\sigma$	3.0e-15
40	$e^- + BV^+ \rightarrow BV^0$	$\sigma$	3.0e-14	109	$h^+ + PV^- \rightarrow PV^0$	$\sigma$	3.9e-14

the generation and recombination term for species  $x$ , and  $D_x$  and  $\mu_x$  are the diffusivity and mobility coefficients for species  $x$ .

The generation/recombination terms  $R_n, R_p, R_{Y_i}, i = 1, \dots, N$  are a sum of source terms arising from the defect reactions. We are primarily interested in carrier-defect reactions such as  $X^m \rightarrow X^{m+1} + e^-$  that contribute a source term of the form

$$R_{X^{m+1}} = \sigma A X^m \exp\left(\frac{\Delta E}{kT}\right), \quad (9)$$

where  $\sigma$  is the reaction cross-section,  $A$  is a constant,  $\Delta E$  is the activation energy,  $k$  is Boltzmann's constant and  $T$  is the lattice temperature. Here  $X$  represents a defect species and superscripts denote charge state. The corresponding source term for the capture reaction  $X^{m+1} + e^- \rightarrow X^m$  has the same form as (9), but with zero activation energy. Similar reactions for release and capture of holes  $h^+$  are also included. For the problem of interest, there are a total of 84 carrier-defect reactions among 35 defect species. A few of these reactions along with their activation energy and cross-section values are summarized in Table 1.

Equations (5-8) are discretized in Charon using a Galerkin finite-element method with two-dimensional bilinear basis functions on quadrangle mesh cells and streamline upwind Petrov-Galerkin (SUPG) stabilization [10, 11]. To keep the problem size reasonable, we chose only to simulate a pseudo one-dimensional vertical strip (9x0.1 micron) through the BJT as shown in Fig. 1(b) — a full two-dimensional simulation would require about a week of computing time on 1000 processors. Dirichlet boundary conditions for the electric potential are applied at each end of the strip, representing the emitter and collector contacts, and also at the emitter-base junction to represent the base contact. The resulting ordinary differential equations are then integrated forward in time using Rythmos, as discussed in the previous section. Forward mode AD in Sacado is used to differentiate the finite-element residual equations: we used the approach described in Sect. 2 to compute the Jacobian and mass matrices for the implicit time integration methods, as well as to compute the analytic derivatives with respect to reaction cross-sections and activation energies for each time step, as required for transient sensitivity analysis.

For comparison to experimental data, the electric current at the base contact is computed as the net carrier flux through the contact and supplies the observation function  $g$ . This calculation naturally decomposes into a set of element computations that can be differentiated via AD in

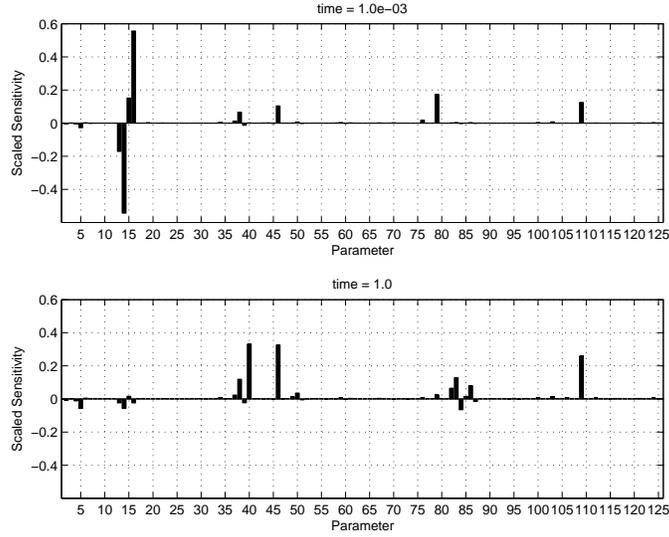
a manner similar to that discussed in Sect. 2 to compute the requisite partial derivatives in (4) for sensitivity analysis.

## 6 Analysis of a Radiation Damaged BJT

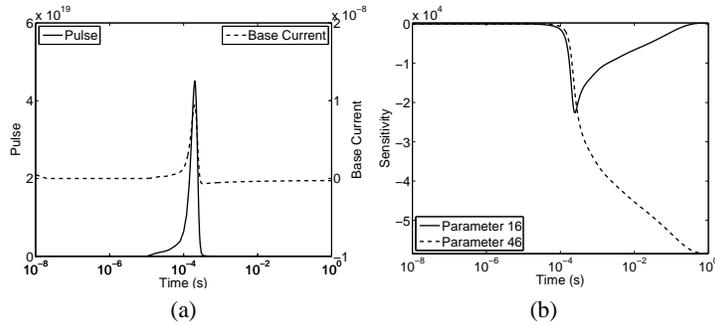
There is significant uncertainty in the defect reaction cross-section and activation-energy parameters that can be reduced by calibrating the computational simulation against (existing) experimental data. To this end, we applied the transient sensitivity method discussed in Sect. 4 to the BJT model from Sect. 5 to compute sensitivities of the electric current at the base contact with respect to all 126 defect reaction parameters, for later use in a derivative-based optimization method to calibrate the model. Dirichlet boundary conditions for the electric potential  $\psi$  are applied at all three contacts, with values of  $-0.589$  (emitter),  $0$  (base) and  $10.21$  (collector). Zero Dirichlet boundary conditions are also applied at the emitter and collector contacts for vacancies, and silicon and boron interstitials. Natural boundary conditions for carriers and all other defect species are applied throughout the boundary. A radiation pulse is simulated by applying a transient source term for generation of Frenkel pairs and electron/hole densities (ionization), as shown in Fig. 3(a). We ran the transient sensitivity calculation over a time interval of  $[0, 1]$ , using Rythmos's adaptive step-size, variable-order BDF method with an initial time step size of  $10^{-8}$  and relative and absolute error tolerances of  $10^{-3}$  and  $10^{-6}$  respectively. The variable-order method was restricted to a fixed order of 1 (backward-Euler method) because Charon exhibited unphysical oscillations with higher-order methods that currently we have not been able to eliminate. The implicit time step equations were solved by NOX using an undamped Newton method with a weighted root-mean-square update-norm tolerance of  $10^{-4}$ . The Newton and sensitivity linear systems were solved by AztecOO using preconditioned GMRES with a tolerance of  $10^{-9}$  (Newton) and  $10^{-12}$  (sensitivity) and Ifpack's RILU(2) preconditioner with one level of overlap. The calculation was run on Sandia's Thunderbird cluster using 32 processors with a discretization of 2770 mesh nodes and 39 unknowns per node (108,030 total unknowns). Scaled sensitivities of the base current with respect to all parameters at early and late times after the radiation pulse are shown in Fig. 2, along with transient sweeps of two of the dominant sensitivities in Fig. 3(b). For each parameter, the scaled sensitivity is given by  $(p/I)(dI/dp)$  where  $p$  is the parameter value,  $I$  is the (base) current, and  $dI/dp$  is the transient sensitivity. A simulation without sensitivities but with identical configuration otherwise requires approximately 105 minutes of computing time, whereas the transient sensitivity calculation for all 126 sensitivities took approximately 931 minutes. Note that because the forward sensitivity solver currently does not implement error control for the sensitivities (as described in Sect. 4), we found by trial and error the tighter  $10^{-12}$  linear solver tolerance was necessary to compute the sensitivities stably.

The primary goal for computing these sensitivities is for later use in a derivative-based optimization method for model calibration. However the relative sensitivities displayed in Fig. 2 also provide important qualitative information by clearly demonstrating which are the dominant parameters and that only a small fraction of the 126 parameters have non-trivial sensitivities. This suggests that an optimization over the 10-15 dominant parameters would likely be just as successful as over the full set, reducing the cost of the model calibration. It also suggests the physics associated with these parameters would be a good target if refinement of the computational model proved necessary.

The typical approach at Sandia for obtaining this sensitivity information is through non-invasive finite-difference methods. However given the small magnitudes of many of the parameters (see Table 1), it is unclear *a priori* what reasonable perturbation sizes would be. We



**Fig. 2.** Scaled transient base current sensitivities at early and late times of the BJT device with respect to the cross-section and activation energy parameters. Sensitivities are scaled to  $(p/I)(dI/dp)$  where  $p$  is the parameter value,  $I$  is the base current, and  $dI/dp$  is the unscaled sensitivity.



**Fig. 3.** (a) Frenkel pair (vacancies and silicon interstitials) and ionization (electron/hole) density source term simulating a radiation pulse (solid curve) and resulting base current (dashed curve). (b) Transient history of (unscaled) sensitivities 16 and 46 from Fig. 2. Unscaled sensitivities are shown because the current  $I$  passes through zero creating a singularity in the scaled sensitivity.

compared computing sensitivities using first-order finite differencing to the direct method in Rythmos and found, not surprisingly, that the Rythmos approach was much faster and more robust. In Table 2, the magnitude of the relative difference in the base current sensitivity with respect to parameter 16 between Rythmos and first-order finite differencing is shown at several

**Table 2.** Magnitude of relative difference in base current sensitivities between Rythmos and first-order finite differencing, at several times and with several finite-difference perturbation sizes  $\epsilon$ , for parameter 16. Here  $\epsilon$  is the relative perturbation size, the absolute perturbation size is  $\epsilon|p|$ , where  $p$  is the value of the parameter (2.4e-14). The absolute difference in all sensitivities is of the order  $10^3$  to  $10^4$ .

Time	$\epsilon = 10^{-0}$	$\epsilon = 10^{-1}$	$\epsilon = 10^{-2}$	$\epsilon = 10^{-3}$	$\epsilon = 10^{-4}$	$\epsilon = 10^{-5}$	$\epsilon = 10^{-6}$
$10^{-4}$	2.7769	0.2219	0.2425	0.3137	0.3143	0.3179	0.3539
$10^{-3}$	0.0888	0.0218	0.0094	0.0118	0.0123	0.0123	0.0124
$10^{-2}$	0.0659	0.0433	0.0520	0.0625	0.0979	0.4599	4.0786
$10^{-1}$	0.0971	0.2159	0.0392	0.0543	0.0528	0.0501	0.0573
$10^{-0}$	0.2724	0.0766	0.1288	0.1602	0.1647	0.1673	0.0681

times for several relative finite-difference perturbation sizes. Generally speaking, the finite-difference value is not terribly sensitive to the perturbation size, but there is no clear single choice that would yield good accuracy for all time points. The difficulty with computing these sensitivities using finite differencing is that parameter perturbations induce variations in time-step sizes that add noise to the sensitivity calculation. This noise can be reduced by tightening the time integrator error tolerances, but this may come at considerable additional computational cost. Clearly computing sensitivities in this way is hard to make robust, which can be critical when embedded in a transient optimization calculation. Moreover, computing sensitivities by finite differencing for this problem is drastically more expensive. Computing all 126 sensitivities via first-order finite differences would take roughly 13,000 minutes (about 9 days) of computing time on 32 processors, compared to 931 minutes using the direct approach in Rythmos. There are three reasons for this difference in cost, all stemming from the fact that each finite-difference calculation requires a full time integration: all of the sensitivities during the early portion of the time integration are zero (which require no work for the sensitivity linear solves), because the sensitivity equations are linear, they only require one linear solve per time step instead of a full Newton solve, and finally the sensitivity linear solves typically require significantly fewer linear solver iterations than the Newton linear solves (currently it is unclear why this is the case).

## 7 Concluding Remarks

We have described the transient sensitivity analysis of a computational simulation of a bipolar junction transistor subject to radiation damage, work that is a step toward a full transient optimization for model calibration. The combination of AD, as implemented in the new Trilinos package Sacado, and the forward sensitivity method in the Trilinos time integration package Rythmos provided efficiency and robustness.

In the future we plan to embed these sensitivity calculations in transient optimization algorithms (provided by MOOCHO, another new Trilinos package) for full model calibration and parameter estimation. For this to succeed, controlling the accuracy of the sensitivity computations is critical; such control is virtually impossible with finite differencing. The next step is to

implement full error control on the sensitivity equations. Applying the error control strategies already in Rythmos to the sensitivity equations should be straightforward.

Typically for an optimization over a parameter space of the size studied here (126), one would expect an adjoint sensitivity approach using reverse-mode AD to be more efficient. While Sacado does provide a reverse-mode capability, this approach would also require an adjoint-enabled time integrator in Rythmos, which has not yet been completely implemented. In the future we do plan to implement adjoint sensitivities in Rythmos, leveraging Sacado for local adjoint sensitivities of the model to further speed up the model calibration problem.

## References

1. Trilinos packages Sacado, Rythmos, NOX, Thyra, Stratimikos, AztecOO and Ifpack are available at the Trilinos web site <http://trilinos.sandia.gov>
2. Abrahams, D., Gurtovoy, A.: C++ Template Metaprogramming. Addison-Wesley, Boston (2005)
3. Aubert, P., Di Césaré, N., Pironneau, O.: Automatic differentiation in C++ using expression templates and application to a flow control problem. *Computing and Visualisation in Sciences* **3**, 197–208 (2001)
4. Bartlett, R.A., Gay, D.M., Phipps, E.T.: Automatic differentiation of C++ codes for large-scale scientific computing. In: V.N. Alexandrov, G.D. van Albada, P.M.A. Sloot, J. Dongarra (eds.) *Computational Science – ICCS 2006, Lecture Notes in Computer Science*, vol. 3994, pp. 525–532. Springer, Heidelberg (2006)
5. Feehery, W.F., Tolsma, J.E., Barton, P.I.: Efficient sensitivity analysis of large-scale differential-algebraic systems. *Appl. Numer. Math.* **25**(1), 41–54 (1997)
6. Gay, D.M.: Semiautomatic differentiation for efficient gradient computations. In: H.M. Bücker, G. Corliss, P. Hovland, U. Naumann, B. Norris (eds.) *Automatic Differentiation: Applications, Theory, and Tools, Lecture Notes in Computational Science and Engineering*. Springer (2005)
7. Hennigan, G.L., Hoekstra, R.J., Castro, J.P., Fixel, D.A., Shadid, J.N.: Simulation of neutron radiation damage in silicon semiconductor devices. Tech. Rep. SAND2007-7157, Sandia National Laboratories (2007)
8. Heroux, M., Bartlett, R., Howle, V., Hoekstra, R., Hu, J., Kolda, T., Lehoucq, R., Long, K., Pawlowski, R., Phipps, E., Salinger, A., Thornquist, H., Tuminaro, R., Willenbring, J., Williams, A., Stanley, K.: An overview of the Trilinos package. *ACM Trans. Math. Softw.* **31**(3) (2005)
9. Hindmarsh, A.C., Brown, P.N., Grant, K.E., Lee, S.L., Serban, R., Shumaker, D.E., Woodward, C.S.: Sundials: Suite of nonlinear and differential/algebraic equation solvers. *ACM Trans. Math. Softw.* **31**(3), 363–396 (2005)
10. Hughes, T.J.R., Franca, L.P., Balestra, M.: A new finite element formulation for computational fluid dynamics: V. Cicumventing the Babuska-Brezzi condition: A stable Petrov-Galerkin formulation of the Stokes problem accomodating equal order interpolation. *Computer Methods in Applied Mechanics and Engineering* **59**, 85–99 (1986)
11. Hughes, T.J.R., Franca, L.P., Hulbert, G.M.: A new finite element formulation for computational fluid dynamics: VIII. the Galerkin/least-squares method for advective-diffusive equations. *Computational Methods Applied Mechanics and Engineering* **73**, 173–189 (1989)
12. Sze, S.M.: *Physics of Semiconductor Devices*, 2nd edn. Wiley & Sons (1981)